实验报告

分布式键值存储系统

题目

设计并实现一个分布式键值（key-value）存储系统，可以是基于磁盘的存储系统，也可以是基于内存的存储系统，可以是主从结构的集中式分布式系统，也可以是 P2P 式的非集中式分布式系统。能够完成基本的读、写、删除等功能，支持缓存、多用户和数据一致性保证。

实验要求

- 基本要求
  - 分布式的键值存储系统，至少在两个节点或者两个进程中测试；
  - 可以是集中式的也可以是非集中式；
  - 能够完成基本的操作如：set, get, del, getall, delall 等 ；
  - 支持多用户同时操作；
  - 至少实现一种面向客户的一致性如单调写；
  - 需要完整的功能测试用例；
  - 涉及到节点通信时须采用 RPC 机制；
- 加分项
  - 增加日志功能
  - 增加用户功能以确保安全性
  - 考虑错误输入

环境配置

python3
rpyc
sqlitedict

具体实现

$client(多个) - master - server(多个) - database$

- client
  - 连接master并运行命令
    - 给出操作指南或者说是展示功能（__doc __返回注释）

```
class Client(object):
"""
commands:
SET key value - set value to key
GET key - get value from key
GETALL - get all key-value
DEL key - delete a key
DELALL - delete all key-value
GETlog - get the log
"""

def try_to_connect(self):
    self.conn = rpyc.connect('localhost', 18861)
    self.id = self.conn.root.get_id()
    return self.id

def run(self):
    try:
        while True:
            command = input('Client %d > ' % self.id)
            if command == 'help':
                print(self.__doc__)
            else:
                msg = self.conn.root.run_command(self.id, command)
                if msg != None:
                    print(msg)
    except KeyboardInterrupt:
        pass
```

  - 用户登录验证与获取用户id

```
users = {}
with open('user.txt') as f:
    for i in f.readlines():
        i = i.split()
        users[i[0]] = i[1]

username = input('Please input your username:')
usercode = input('Please input your code:')
if username in users.keys() and users[username] == usercode:
```

```
    client = Client()
    client_id = client.try_to_connect()
    if client_id == None:
        print('Connection Failed.')
    else:
        print('Welcome to GZY simple distributed key-value database.')
        print('Your client ID is %d.' % client_id)
        print('Enter help for more info if necessary.')
        client.run()
else:
    print('Your username or code has something wrong.please retry.')
```

- server
  - 实现各种功能
    - 实际上就是对数据库的增删修改等操作，还有日志功能。

```
class Server(rpyc.Service):
"""Server"""
def on_connect(self, conn):
        self.db = SqliteDict('./database.sqlite', autocommit=True)
        def exposed_set(self, k, v):
        self.db[k] = v

        def exposed_get(self, k):
        return None if k not in self.db else self.db[k]

        def exposed_rem(self, k):
                if k not in self.db:
                        return
        del self.db[k]

        def exposed_getall(self):
        return [(k, self.db[k]) for k in self.db]

        def exposed_delall(self):
                keys = [k for k in self.db]
                for k in keys:
                        del self.db[k]

        def exposed_writelog(self,msg):
        log.append(msg)

def exposed_getlog(self):
        return log
```

- master
  - 对不同用户的命令进行执行

- 首先对用户命令进行判断

```
def judge_command(cmd):
  if cmd == 'set':
      return 0
  if cmd == 'get' or cmd == 'del':
      return 1
  if cmd == 'getall' or cmd == 'delall':
      return 2
  if cmd == 'getlog':
      return 3
  return 4
```

- 再对不同的命令进行不同的执行

```
def exposed_run_command(self, client_id, clause):
clause = clause.strip().split()
lens = len(clause)
WRONG_MSG = 'Wrong command. Enter help if necessary.'
if lens < 1:
    return WRONG_MSG

command = clause[0].lower()
cmd_type = self.judge_command(command)
if cmd_type == 4:
    return WRONG_MSG

elif cmd_type == 0:  # set
    if lens != 3:
        return WRONG_MSG

    k, v = clause[1], clause[2]
    clients[client_id].root.set(k, v)
    clients[client_id].root.writelog('set '+str(k)+' - '+str(v))

elif cmd_type == 3:
    if lens != 1:
        return WRONG_MSG
    return clients[client_id].root.getlog()

elif cmd_type == 1:  # get or del
    if lens != 2:
        return WRONG_MSG
    k = clause[1]
    if command == 'get':
        msg = clients[client_id].root.get(k)
        clients[client_id].root.writelog('get '+str(k))
        if msg == None:
            return 'Key %s not found.' % k
        else:
```

```
                return msg
        else:
            clients[client_id].root.rem(k)
            clients[client_id].root.writelog('del '+str(k))

    else:  # getall or delall
        if lens != 1:
            return WRONG_MSG
        if command == 'getall':
            clients[client_id].root.writelog('getall')
            return clients[client_id].root.getall()
        else:
            clients[client_id].root.writelog('delall')
            return clients[client_id].root.delall()
```

功能测试

下面对实验要求中的各个要求都加以验证
首先运行

```
python server.py -p 5
```

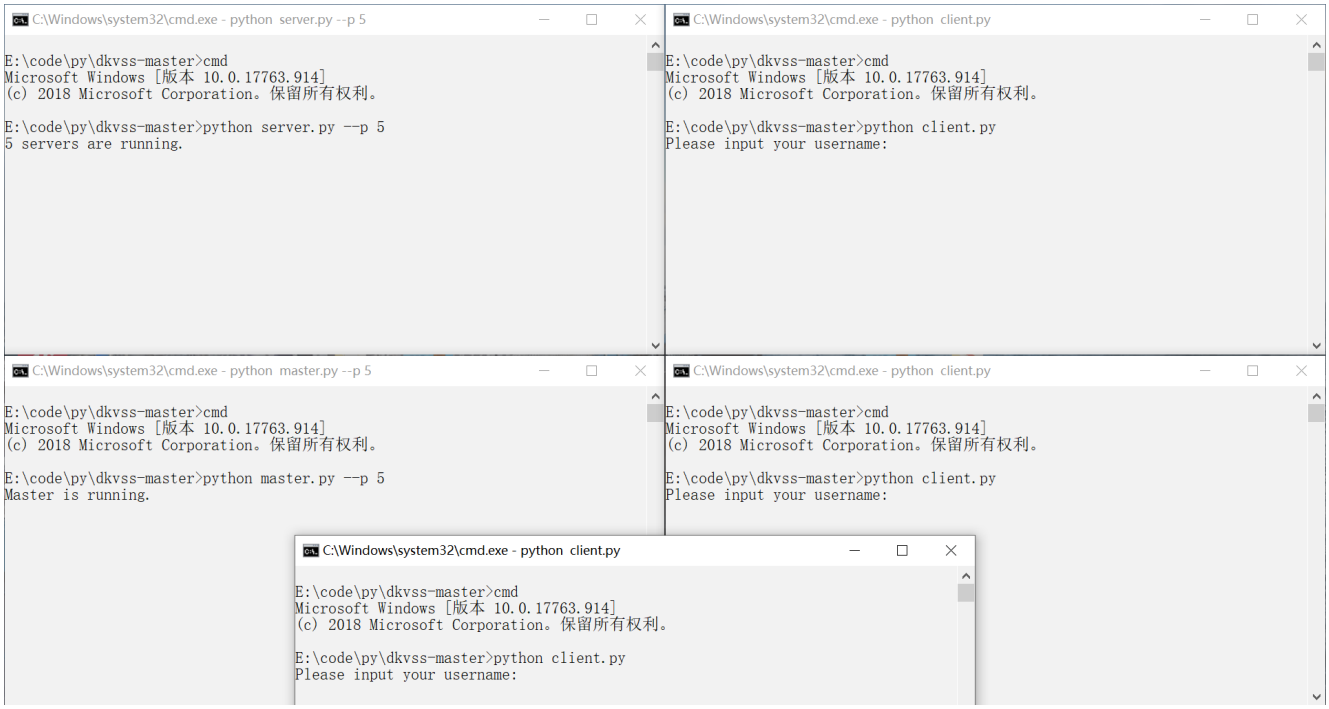
随后运行master，参数不得大于服务器数量

```
python master.py -p 5
```

再打开一个或多个窗口运行client，个数也不得超过master的参数

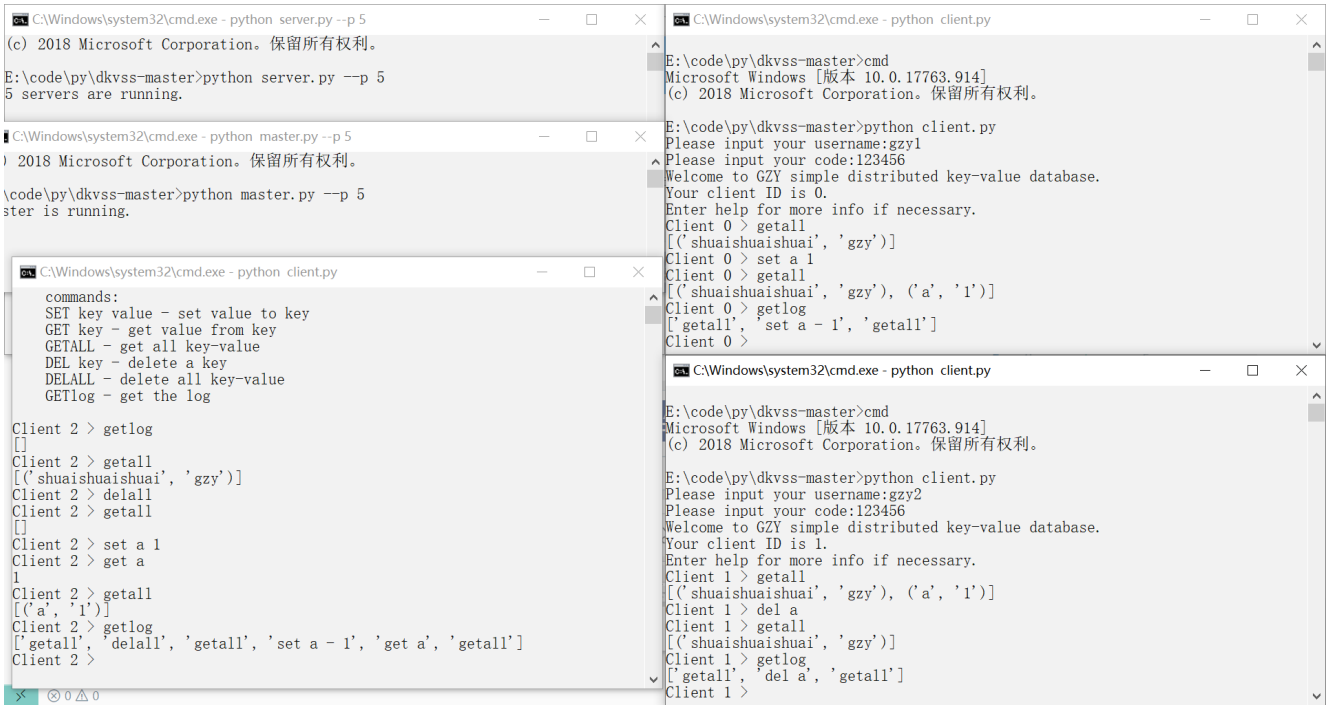
```
python client.py
```

效果如下：

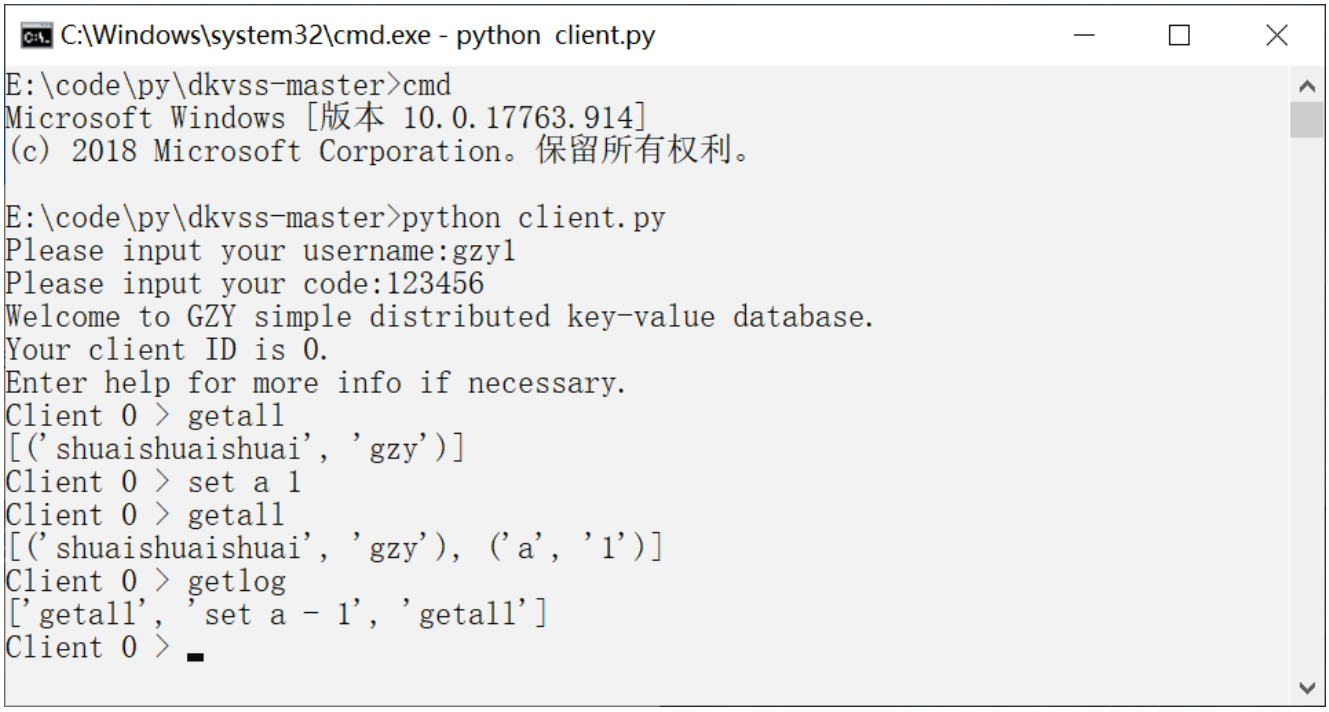


如图为一个server，一个master，三个client的示例。

现在登录client0，1，2，验证各种功能。

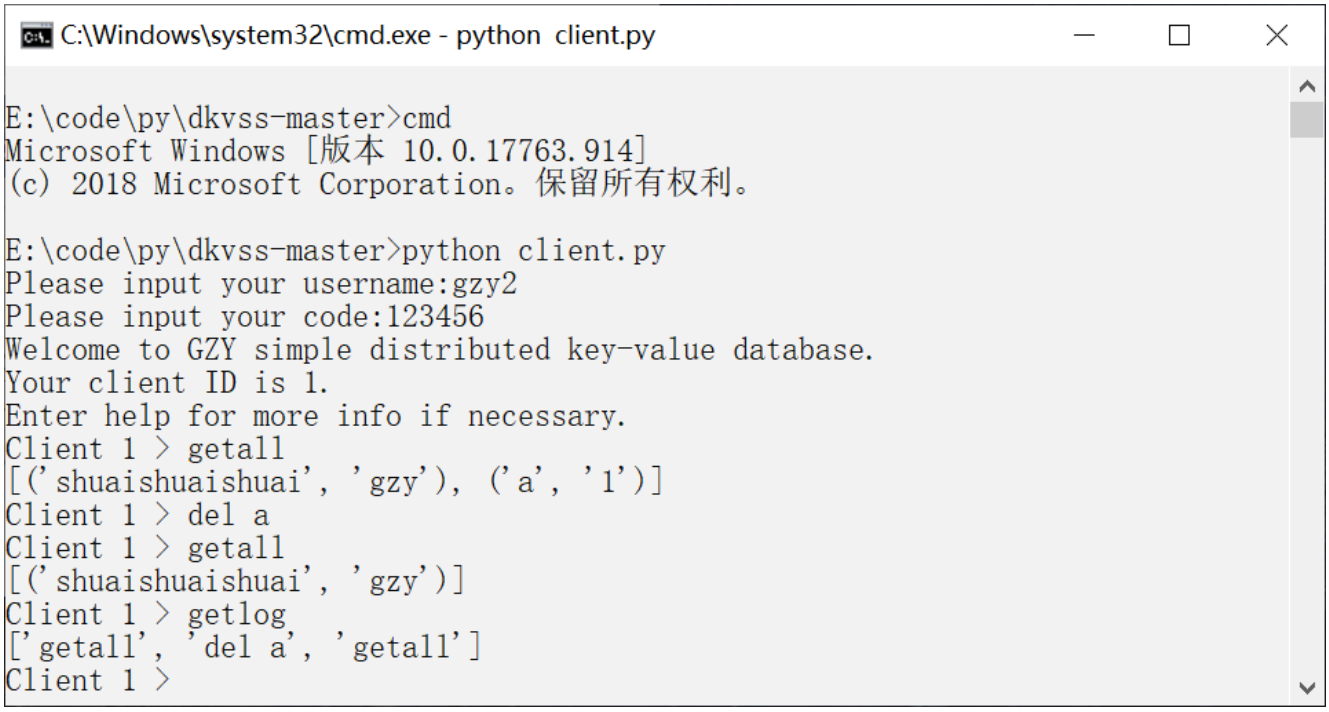


上图显示多个进程，支持多用户合作。

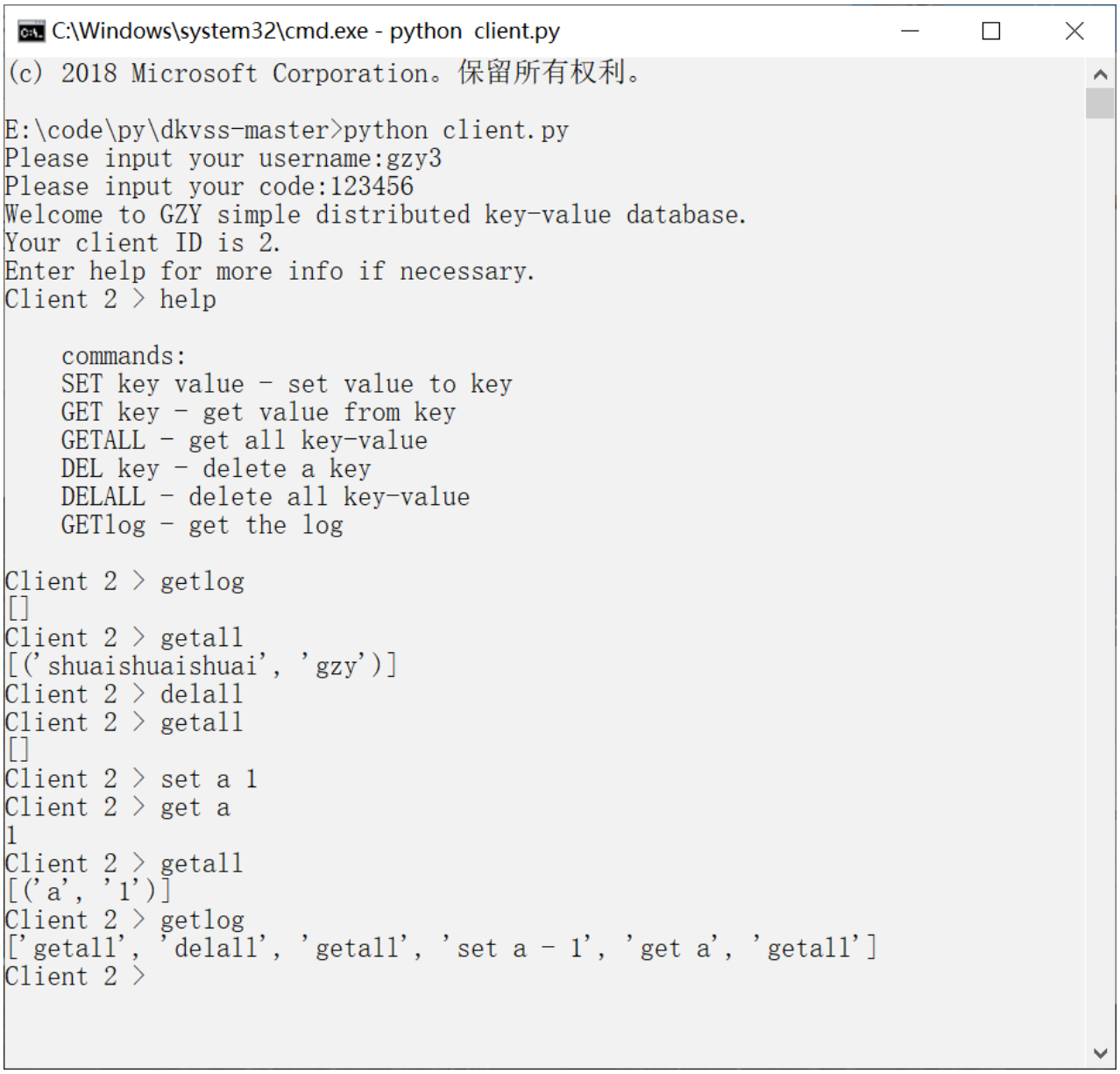


在第一个用户登录后进行查看，增加set，查看，查看日志的操作

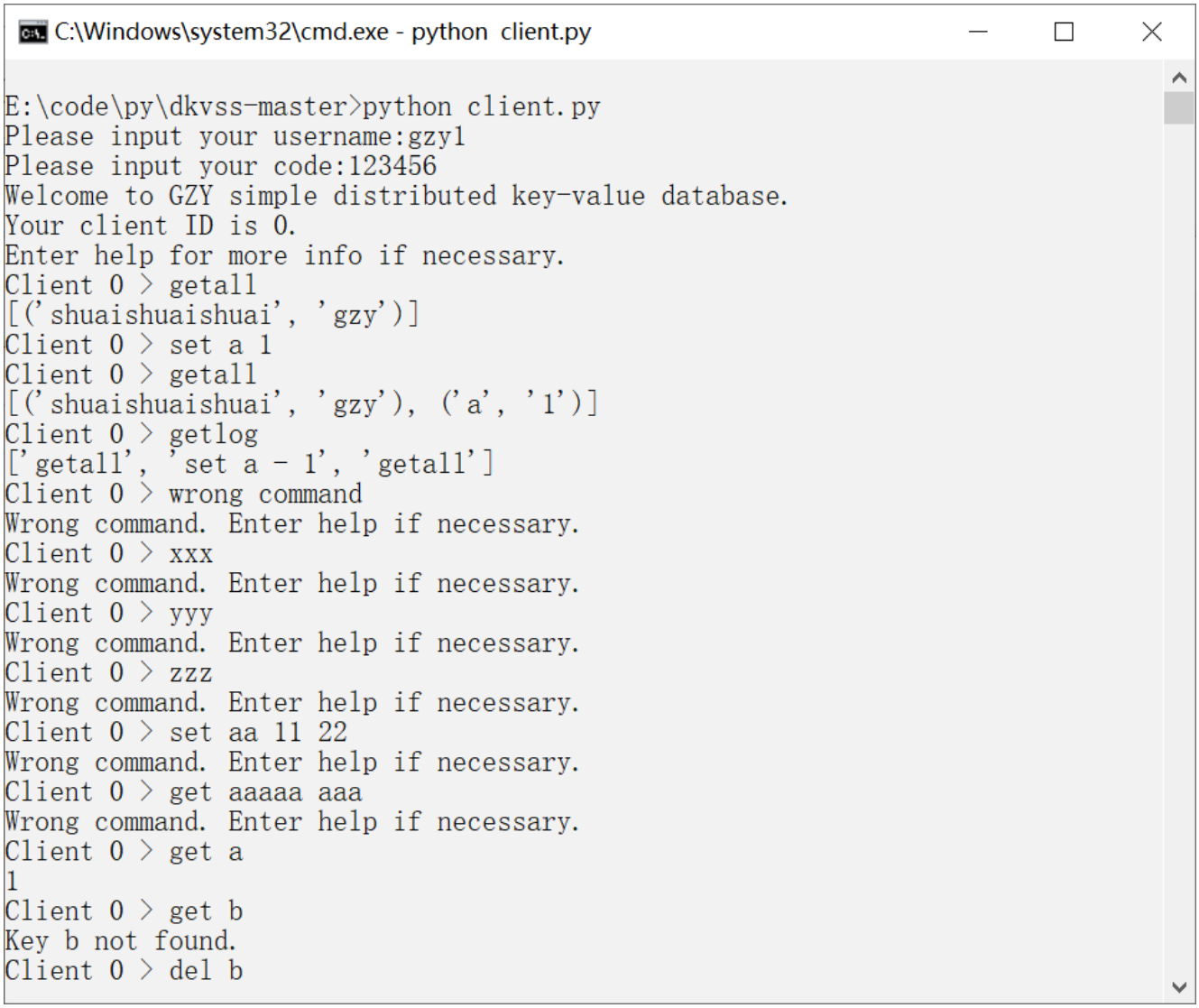
注意：每个用户只能查看自己的操作日志，而不能查看他人的操作。



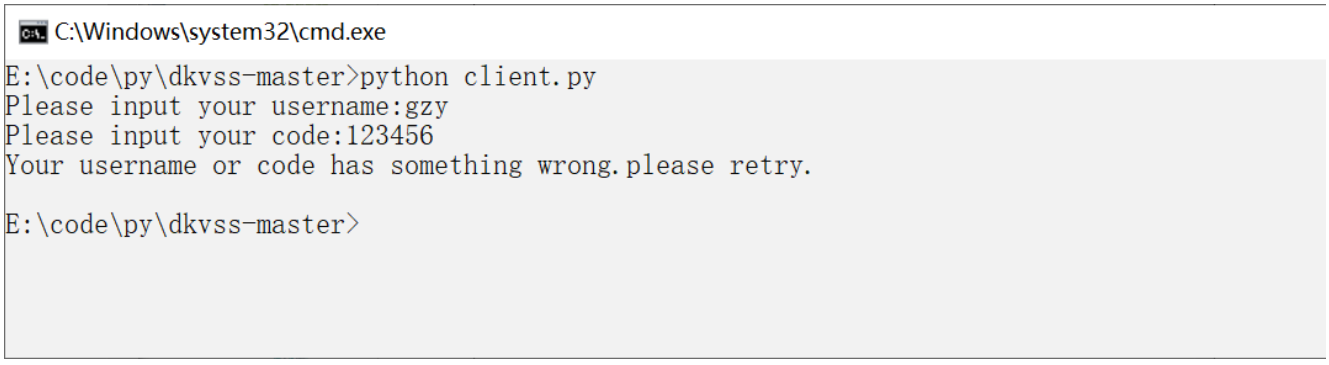
在第一个用户增加后立马在第二个查看发现已经是被修改了，体现了面向用户的一致性，同样测试删除del操作，正常删除，然后查看日志，对比client 0的用户日志可以明显看到并没有查看到client 0 的操作日志，功能运行正常。



在client 2 中输入help查看指南，测试delall，get，getlog等功能，功能正确执行。



在client 0 中输入错误命令，包括完全错误，部分错误，以及测试get 不在数据库中的key，程序的反应均与用户良好。



输入错误的用户名以及密码将会提示错误并让用户重试。
综上，经过了这么多测试，已经验证了该系统具备以下功能(即实验要求的功能以及加分项)：

- 基本要求
  - 分布式的键值存储系统，至少在两个节点或者两个进程中测试；
  - 可以是集中式的也可以是非集中式；
  - 能够完成基本的操作如：set, get, del, getall, delall 等；
  - 支持多用户同时操作；
  - 至少实现一种面向客户的一致性如单调写；
  - 需要完整的功能测试用例；

    - 涉及到节点通信时须采用 RPC 机制；
- 加分项
    - 增加日志功能
    - 增加用户功能以确保安全性
    - 考虑错误输入

总结

由于老师说可以参考网上的代码，所以我在参考了网上一些键值分布式存储系统的实现后实现了简单的分布式键值存储系统，基本上每个功能我都进行了自己的实现，特别是加分项，即日志功能，用户登录功能，更多的考虑用户输入，以及各种set, get, del, getall, delall功能我都进行了魔改，也可以看到实现了许多独特的功能，也算是交出了一份属于自己的答卷。